Classification_KNN

October 20, 2025

[1]:
```
# Import numpy for numerical operations and array handling
import numpy as np

# Import pandas for data manipulation and analysis
import pandas as pd

# Import the Iris dataset from scikit-learn's built-in datasets
from sklearn.datasets import load_iris

# Import train_test_split to split data into training and testing sets
from sklearn.model_selection import train_test_split

# Import StandardScaler to normalize feature values for KNN
from sklearn.preprocessing import StandardScaler

# Import KNeighborsClassifier to build the KNN model
from sklearn.neighbors import KNeighborsClassifier

# Import metrics for model evaluation: confusion matrix, accuracy, precision,␣
 ↪recall, and a full report
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
 ↪recall_score, classification_report
```

[2]:
```
# Load the Iris dataset into memory
data = load_iris()

# Extract the feature matrix (X) and target vector (y)
X = data.data  # Features: sepal length, sepal width, petal length, petal width
y = data.target  # Target: species encoded as integers (0, 1, 2)

# Convert the data to a pandas DataFrame for easier inspection and manipulation
iris_df = pd.DataFrame(X, columns=data.feature_names)
iris_df['species'] = y  # Add the target as a new column

# Display the first five rows to get a sense of the data structure and values
iris_df.head()
```

[2]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species
0        0
1        0
2        0
3        0
4        0
```

[3]:
```
# Split the dataset into training and testing subsets
# test_size=0.3 means 30% of the data will be used for testing, 70% for training
# random_state ensures reproducibility of the split
# stratify=y ensures the class distribution is preserved in both sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
```

[4]:
```
# Initialize the StandardScaler to normalize features
scaler = StandardScaler()

# Fit the scaler on the training data and transform it
# This step computes the mean and std on the training set, then scales it
X_train_scaled = scaler.fit_transform(X_train)

# Use the same scaler to transform the test data (do NOT fit again)
# This ensures the test data is scaled using the same parameters as the␣
 ↪training data
X_test_scaled = scaler.transform(X_test)
```

[5]:
```
# Initialize the KNN classifier with k=3 neighbors (a common choice for Iris)
knn = KNeighborsClassifier(n_neighbors=3)

# Train the KNN model using the scaled training data and corresponding labels
knn.fit(X_train_scaled, y_train)
```

[5]:
```
KNeighborsClassifier(n_neighbors=3)
```

[6]:
```
# Use the trained KNN model to predict the species of the test set samples
y_pred = knn.predict(X_test_scaled)
```

[7]:
```
# Compute the confusion matrix to see how well the model classified each class
cm = confusion_matrix(y_test, y_pred)
```

```
print('Confusion Matrix:')
print(cm)

# Calculate the overall accuracy: proportion of correct predictions
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Calculate precision: how many selected items are relevant (macro for␣
 ↪multiclass)
precision = precision_score(y_test, y_pred, average='macro')
print(f'Precision (macro): {precision:.2f}')

# Calculate recall: how many relevant items are selected (macro for multiclass)
recall = recall_score(y_test, y_pred, average='macro')
print(f'Recall (macro): {recall:.2f}')

# Print a detailed classification report with precision, recall, f1-score for␣
 ↪each class
print('\nClassification Report:')
print(classification_report(y_test, y_pred, target_names=data.target_names))
```

```
Confusion Matrix:
[[15  0  0]
 [ 0 15  0]
 [ 0  4 11]]
Accuracy: 0.91
Precision (macro): 0.93
Recall (macro): 0.91

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.79      1.00      0.88        15
   virginica       1.00      0.73      0.85        15

    accuracy                           0.91        45
   macro avg       0.93      0.91      0.91        45
weighted avg       0.93      0.91      0.91        45
```

[ ]: